
Jet Propulsion Laboratory
California Institute of Technology

# MAL HTTP Transport
# JPL Status

*Adrian Tinio*

*11/6/2017*

# Task Overview

**Objective**: Provide an **independent** implementation of

1. HTTP transport binding to the MO MAL
2. XML Encoding of MAL messages

As specified in the proposed draft recommended standard CCSDS 524.3-R-0

# Task Breakdown

- MAL Header Encoding
  - Implement encoder/decoder of MAL message header to MAL HTTP header
- MAL Body Encoding
  - Implement encoder/decoder of MAL message body using MAL XML Encoding scheme
- MAL HTTP Transport Interface Binding
  - Implement http/https transport

# Schedule/Status

| Task | Start | End | Status |
|---|---|---|---|
| Initial Task Orientation | 3/20/2017 | 5/5/2017 | Complete |
| MAL Header Encoding Implementation | 5/5/2017 | 5/30/2017 | Complete |
| MAL Body Encoding Implementation | 6/2/2017 | 7/14/2017 | Complete |
| MAL HTTP Transport Implementation | 7/14/2017 | 8/30/2017 | Complete |
| Internal integrated testing | 8/10/2017 | 8/30/2017 | Complete |
| Interoperability Testing* | 11/6/2017 | 11/17/2017 | Tentative schedule with TPZ VEGA |

* Interoperability testing to verify implementation delayed due to unavailability of test personnel

# Task Metrics

| Task Item | Level Of Effort (work weeks) | Lines Of Code | Code Coverage | Comment |
|---|---|---|---|---|
| Initial Task Orientation | 3 | N/A | | Familiarizing with task, reading blue books, white books, green books |
| MAL Header Encoding Implementation | 1.5 | core: 534<br>test: 302 | 92% | Relatively straightforward. Mapping is well defined. |
| MAL Body Encoding Implementation | 6 | core: 1761<br>test: 5075 | 90% | Implements MAL XML encoding |
| MAL HTTP Transport Implementation | 7 | core: 1582<br>test: 2688 | 92% | Supports both http and https |
| | | | | |

# Findings

**Waiting on interoperability testing to expose issues with implementation resulting from misinterpretation of standard, if any.

- During the course of the task, the JPL team needed numerous clarifications on the standard and/or the use of existing software

- A weekly log was kept to record questions:
  - 41 questions; most resolved with input from experts

- *Sampling of open questions in the following slides, with details in backup slides*

# Potential RID (Documentation) 1/2

| Item | Doc-Section | Description |
|---|---|---|
| Support for QoS Level | CCSDS 524.3 4.2.2 | Description not clear on how to implement method call for supportedQOSLevel |
| Sending Http-Response Status in Http-Request Header | CCSDS 524.3 3.2.3.2 | Unclear regarding use of http status code 200 OK for INVOKE_RESPONSE |
| Logic in converting MAL Header to HTTP header field “URI-TO” | CCSDS 524.3 3.5.4 | Unclear how to go from HTTP header URI-TO back to MAL Header field. |

# Potential RID (Software Library) 2/2

| Item | Doc-Section | Description |
|---|---|---|
| Outdated code in MAL-implementation library | | MAL-Transport-Generic library performs encoding on JAXB XML objects. Encoding of JAXB XML objects not in standard. |
| FineTime Interpretation Details | CCSDS 521.0-B-2 4.3.17 | FineTime is specified to have up to picosecond resolution. This seemed to have changed later on to nanosecond. |
| Default constructors for generated data types | CCSDS 521.0-B-2 4.4.5 | Code generator generates default constructors that intialize fields to null fields. Problematic with data types with fields that are not nullable as specified in standard. |
| Missing Transmit Acknowledge Primitive (API) | CCSDS 524.3 4.4.6 | Existing Java MAL implementation-library does not have Transmit_Ack primitive |

# To-Do

- Inter-operability testing to be conducted by TPZ-VEGA personnel.
- Address issues uncovered by interops testing
- Provide support with test plan/report documentation, if necessary.
- Release the prototype software as open-source

# Backup

# Support for QoS Level

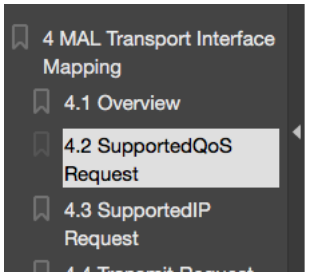
In Section 4.2.2 of MAL Binding to HTTP-Transport & XML-Encoding (CCSDS 524.3-R-0), there is a description on how to implement a method call supportedQoSLevel.

SupportedQoSLevel is a mandatory method to be implemented since we are using the MAL-implementation-library.

But the description is not clear on how to implement them.

After asking ESA / ESOC, they suggested to make an RID for this, but the logic was never explained.

Since this is not integral part of the implementation. The implementation is bypassed, i.e. always return true.




**4.2 SUPPORTEDQOS REQUEST**

**4.2.1** The SUPPORTEDQOS request primitive shall be provided.

**4.2.2** Support for the Quality of Service (QoS) levels defined by MAL shall depend on the capabilities of the underlying layer used to convey the HTTP messages.

# Sending Http-Response Status in Http-Request Header

In Section 3.2.3.2, there is a table named “3-4” mapping Http-Response status codes to their respective stages. One of the stages for the code “200 OK” is called “INVOKE_RESPONSE”.

**Table 3-4: Status Codes in Response Message**

| Status Code | Description | Usage |
|---|---|---|
| 200 **OK** | Standard response for successful HTTP requests. | SUBMIT_ACK<br>REQUEST_RESPONSE<br>INVOKE_RESPONSE<br>PROGRESS_ACK<br>REGISTER_ACK<br>PUBLISH_REGISTER_ACK<br>DEREGISTER_ACK<br>PUBLISH_DEREGISTER_ACK |
| 202 **Accepted** | The request has been accepted for processing, but the processing has not been completed. | INVOKE_ACK |

Continued to next slide…

# Sending Http-Response Status in Http-Request Header Cont..

However, “INVOKE_RESPONSE” is a new Http-Request stage which does not require a response status. This is confirmed by Table 3-3 in Section 3.2.2.2. Hence, the status code is ignored.

**3.2.2.2 Requirements**

The mapping of the MAL interaction messages to the corresponding HTTP request response messages, and the HTTP message initiator, shall be as specified in Table 3-3. full description of the relevant interaction types, the MAL interaction stages, and the init given below see reference [1].

**Table 3-3: MAL Interaction to HTTP Message Mapping**

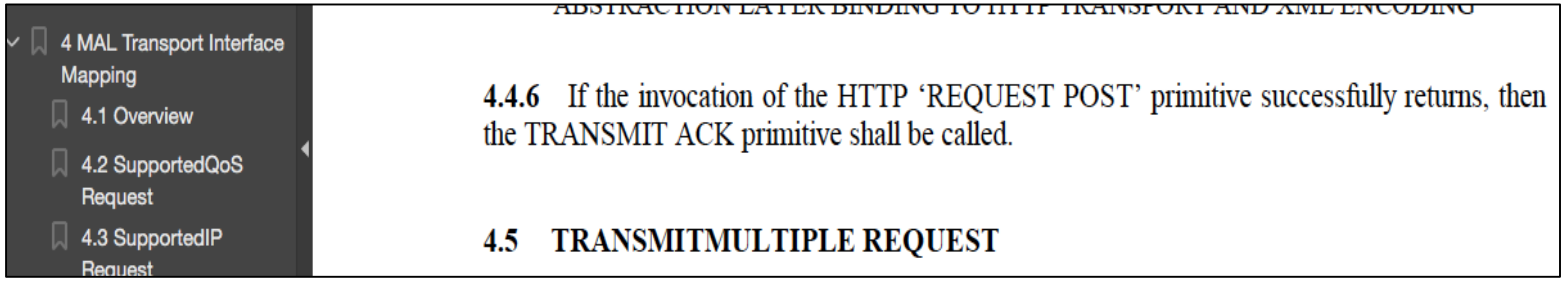
| Interaction Type | MAL Interaction Stage | HTTP Message | |
|---|---|---|---|
| | | Request / Response | Initiator |
| SEND | SEND | Request | Consumer |
| | n/a | Response | Provider |
| SUBMIT | SUBMIT | Request | Consumer |
| | ACK | Response | Provider |
| | ERROR | Response | Provider |
| REQUEST | REQUEST | Request | Consumer |
| | RESPONSE | Response | Provider |
| | ERROR | Response | Provider |
| INVOKE | INVOKE | Request | Consumer |
| | ACK | Response | Provider |
| | ACK_ERROR | Response | Provider |
| | RESPONSE | Request | Provider |
| | n/a | Response | Consumer |
| | RESPONSE_ERROR | Request | Provider |
| | n/a | Response | Consumer |

# Missing Transmit Acknowledge Primitive (API)

In Section 4.4.6, the document says Transmit_Ack Primitive needs to be invoked when an Http-Request is successful.

When it is implemented, MAL-implementation-library does not have such API to be called.

Since this is not the integral part of the implementation, and the assumed purpose of such call is mainly to log it, implementation ignores this instruction.



# Logic in converting MAL Header to Http Header Filed "URI-TO"

In Section 3.5.4, The logic on how to convert one of the MAL Message Header named "URI-TO" to Http Header field is explained. This includes splitting "URI-TO" to 3 different Http Headers: "HOST", "X-MAL-URI-TO", and "REQUEST-TARGET".

There is no explicit explanation on how to reverse them on the receiving end when converting Http message to MAL message.

The implementation assumes that the logic should be reversed.

**3.5.4 URI TO**

**3.5.4.1** If the final destination of the MAL message as specified in the MAL header field 'URI To' coincides with the HTTP destination endpoint, the MAL 'URI To' is mapped as follows:

**3.5.4.1.1** The IP address (or host name) and TCP port number of the MAL header field 'URI To' shall be assigned to 'Host' message header field.

**3.5.4.1.2** If the MAL header field 'URI To' contains a Source Id, then this identifier preceded with a '/' shall be assigned to the 'request-target' field of the MAL HTTP request-line.

**3.5.4.1.3** If the MAL header field 'URI To' does not contain a Source Id, then '/' shall be assigned to the 'request-target' field of the MAL HTTP request-line.

**3.5.4.2** If the HTTP MAL application is being used to route to another MAL node, where the final destination of the MAL message as specified in the MAL header field 'URI To' does not coincide with the HTTP destination endpoint, the MAL 'URI To' is mapped as follows:

**3.5.4.2.1** The MAL header field 'URI To' shall be assigned to 'X-MAL-URI-To' HTTP message header field.

**3.5.4.3** The content of the 'X-MAL-URI-To' HTTP message header field shall be encoded according to the rules defined in reference [9], section 2.

**3.5.4.3.1** The IP address (or host name) and TCP port number of the HTTP destination endpoint shall be assigned to 'Host' message header field.

**3.5.4.3.2** The request-line is implementation-specific.

# FineTime Interpration Details

In MAL data structures, there is a time data type called FineTime. In Section 4.3.17 of MO-MAL book (CCSDS 521.0-B-2), FineTime is an absolute date and time up to picosecond resolution.

When implementing it, there were some issues finding picosecond libraries in Java. After asking to ESA, the definition has been changed to nanosecond. The issue is discussed in this GitHub issue ticket.

# Default Constructors for Generated Data Types

When MAL data types are generated by code generator using the XML templates, default constructors are created. For some data types, especially composite data types (which does not accept null fields), using default constructors will results in exceptions.

However, default constructors are compulsory as the encoder logic is tied to it. Hence, there should be some information for developers not to use them blindly.

Example: Subscription datatype does not allow null fields as explained in blue book.

**4.4.5 SUBSCRIPTION**

The Subscription structure is used when subscribing for updates using the PUBSU interaction pattern. It contains a single identifier that identifies the subscription being define and a set of entities being requested.

| Name | Subscription | | |
|---|---|---|---|
| Extends | Composite | | |
| Short Form Part | 23 | | |
| **Field** | **Type** | **Nullable** | **Comment** |
| subscriptionId | Identifier | No | The identifier of this subscription. |
| entities | List<EntityRequest> | No | The list of entities that are being subscribed for by this identified subscription. |

# Default Constructors for Generated Data Types

Example: Subscription datatype does not allow null fields as explained in blue book. But Subscription code in API_MAL library has default constructor which will initialize fields to null.

**4.4.5 SUBSCRIPTION**

The Subscription structure is used when subscribing for updates using the PUBSU interaction pattern. It contains a single identifier that identifies the subscription being define and a set of entities being requested.

| Name | Subscription | | |
|---|---|---|---|
| Extends | Composite | | |
| Short Form Part | 23 | | |
| **Field** | **Type** | **Nullable** | **Comment** |
| subscriptionId | Identifier | No | The identifier of this subscription. |
| entities | List<EntityRequest> | No | The list of entities that are being subscribed for by this identified subscription. |